Gene-Environment Interaction Analysis using GPU

Daniel Berglund

17 February 2015

Problem and Aim

- Genetic and environmental factors are know to affect the risks of diseases
- Interaction can exist between these factors
- Data amounts are increasing, need for better programs
- GPUs have previously shown good results for gene-gene interaction

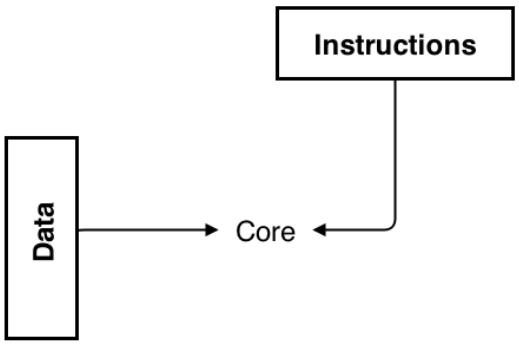
The Data

- The data consists of individuals, their status, environmental factor, SNPs and covariates
- A SNP is an allele in the genome
- The environmental factor is something in the environment, for instance smoking
- Covariates are variables that needs to be adjusted for, for instance age

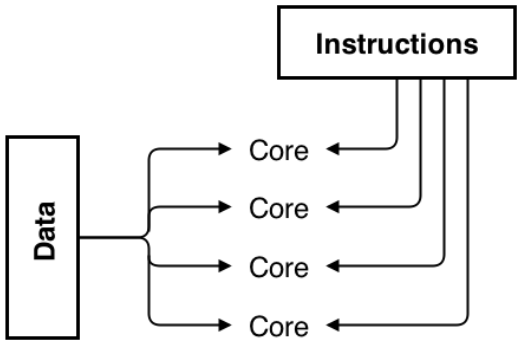
Logistic Regression

- Logistic Regression can be used to model interaction
- Iterative method
- Models probabilities of the outcomes as a linear model
- Models each each gene-environment combination separately
- The model has three variables, one for the SNP, one for the environment and one for the interaction

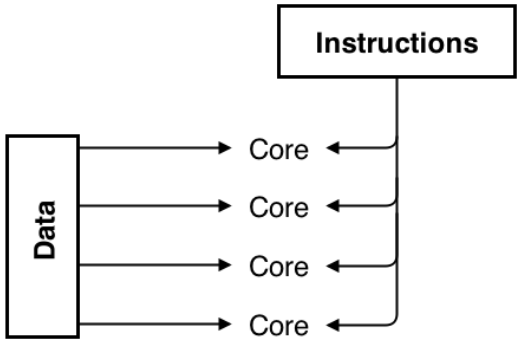
Concurrency Architectures



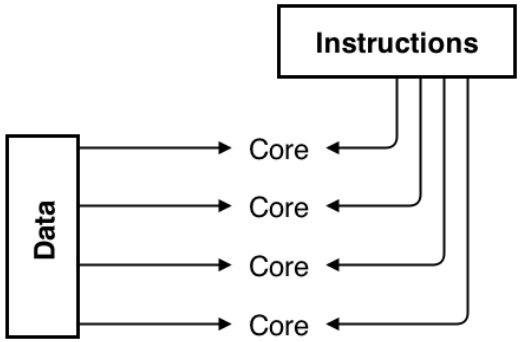
SISD



MISD



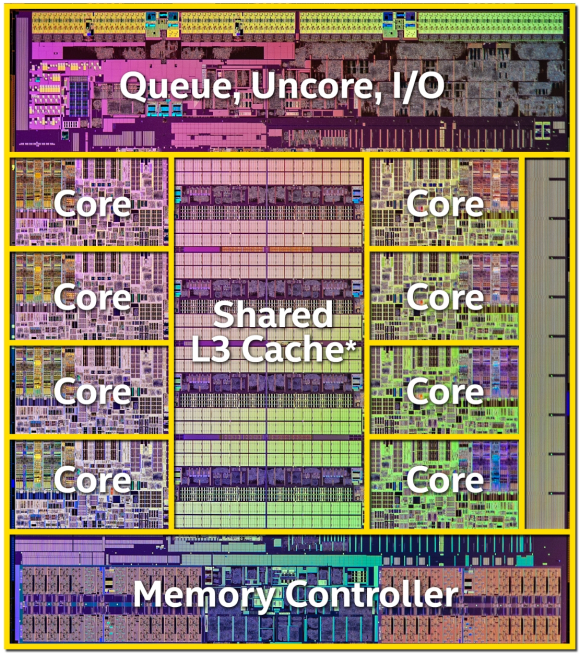
SIMD



MIMD

CPU Architecture

- MIMD
- 8 cores
- Lots of various optimizations
- 20 MB cache
- Similar performance with single and double precision



GPU Architecture

- Separate memory from the normal RAM
- Data has to be explicitly transferred to and from the GPU
- Fewer optimizations than CPU
- Slow but many cores, up to 5000
- Much better performance with single precision than double
- Single instruction, multiple thread(SIMT)

CPU versus GPU

- Pick tool based on problem
- CPU for complex control flows
- GPUs need embarrassingly parallelism
- CPUs are becoming more like GPUs and vice versa
- Xeon Phi, CPU accelerators

CUDA

- For NVIDIA GPUs by NVIDIA
- Supported by various libraries, for instance CUBLAS, Thrust
- Calculations done by kernels

Kernel Example

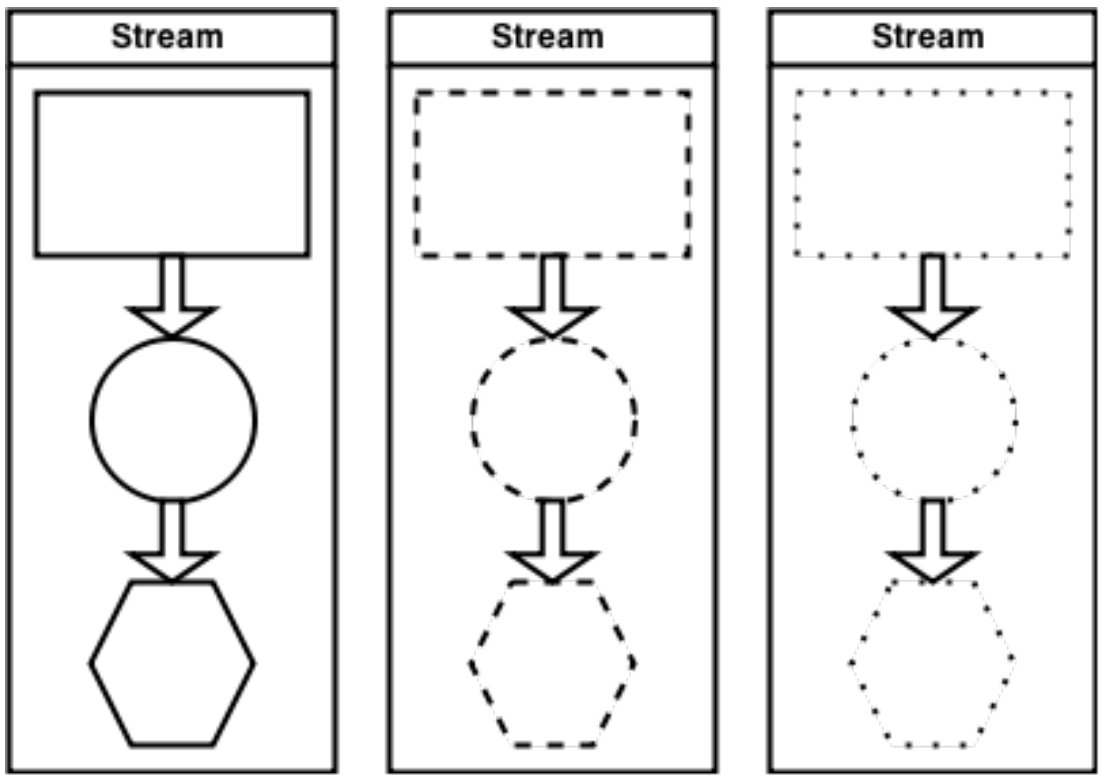
```
__global__ void Add(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

Add<<< N,M >>>(A, B, C);
```

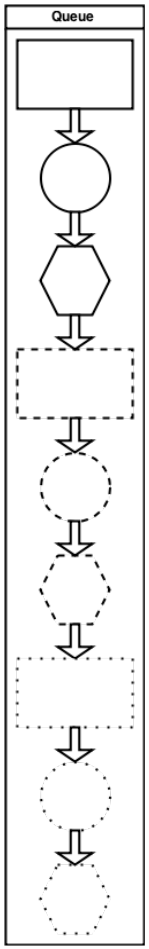
Streams

- Kernels and transfers can be performed on streams
- Transfers and kernels can overlap when using streams

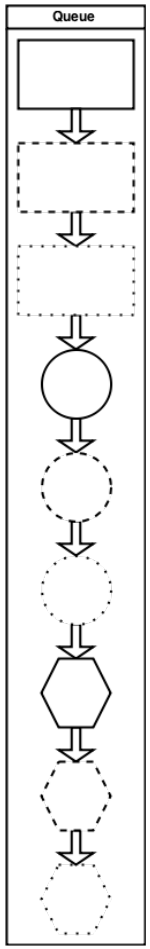
Streams



Streams



Looped over stream



Looped over kernel

Agile Development

- Code in small units, commonly classes
- Unit testing, test the units in isolation
- Mocks, i.e. fake objects
- If a unit is working incorrectly only its corresponding unit test should fail
- Integration tests can be used to test if the units work properly together

Wrappers

- Wrappers are used to “fix” interfaces
- Does not perform the task, delegates it

Wrappers

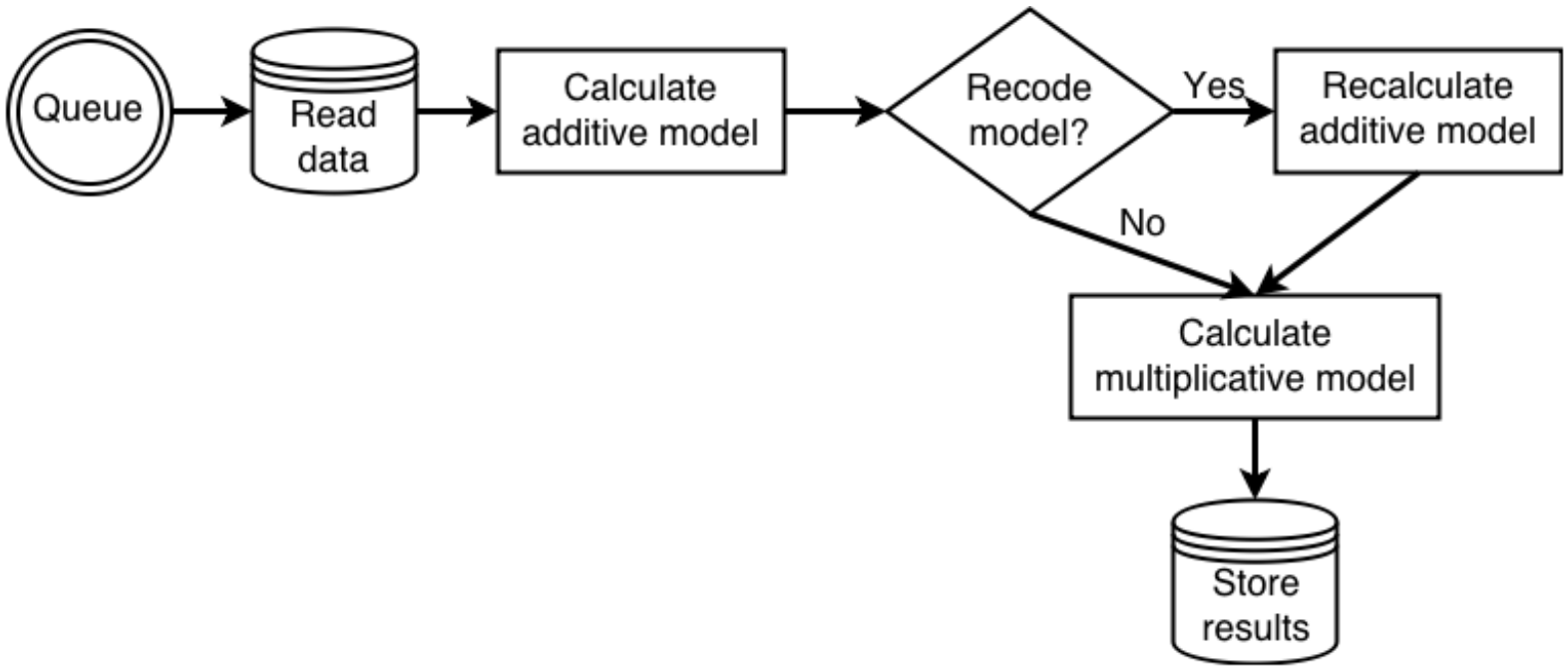
cublasSgemv(*cublasHandle*, *transpose*, *m*, *n*, α, *matrix*, *ld_matrix*, *vector*, *inc_vector*, β, *result*, *inc_result*);

matrixVectorMultiply(*matrix*, *vector*, *result*, α, β);

Structure of Program

- Written in C++
- Libraries used are Google Test/Mock, Boost, CUDA, CUBLAS
- Modularised
- Almost full unit test coverage
- Handles each gene environment combination independently
- Some parts are done using CPU

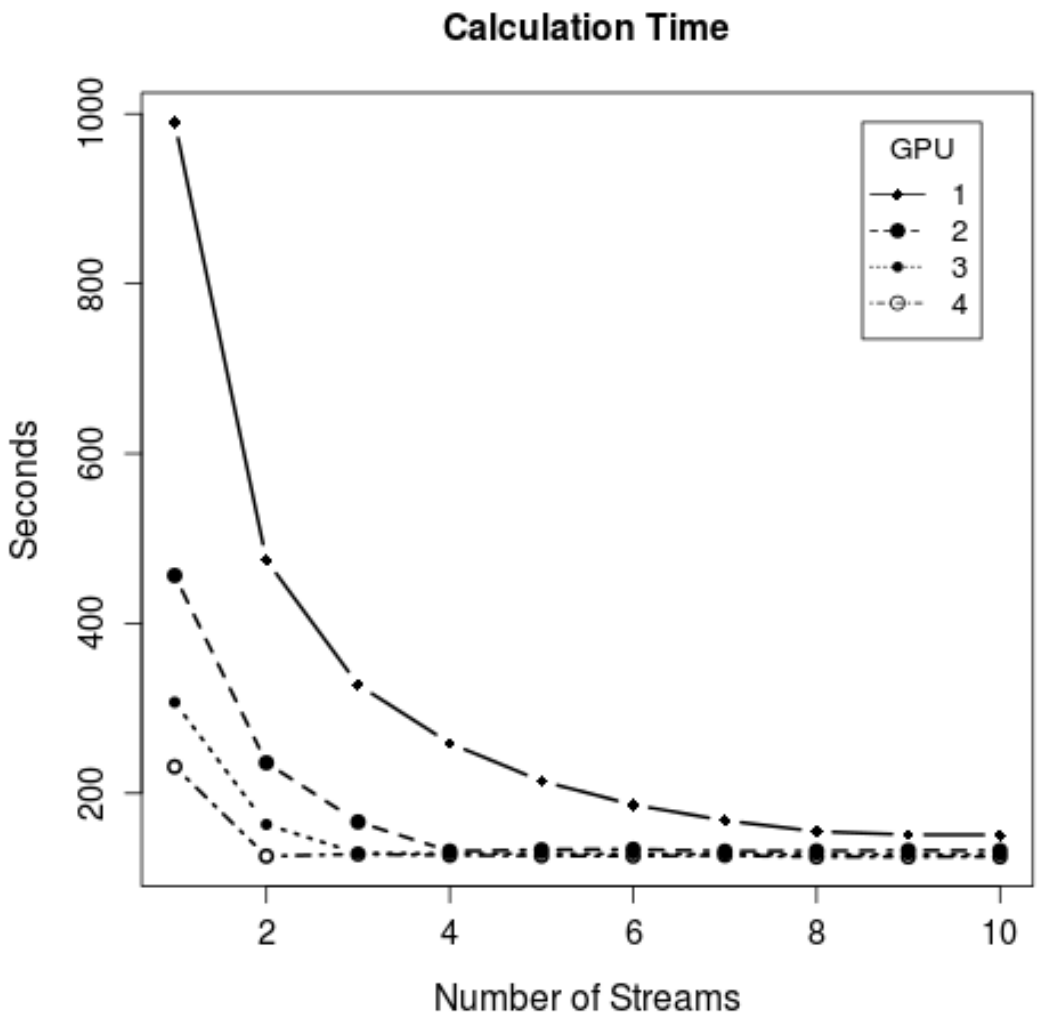
Overview


Queue
Read data
Calculate additive model
Recode model?
Yes
Recalculate additive model
No
Calculate multiplicative model
Store results

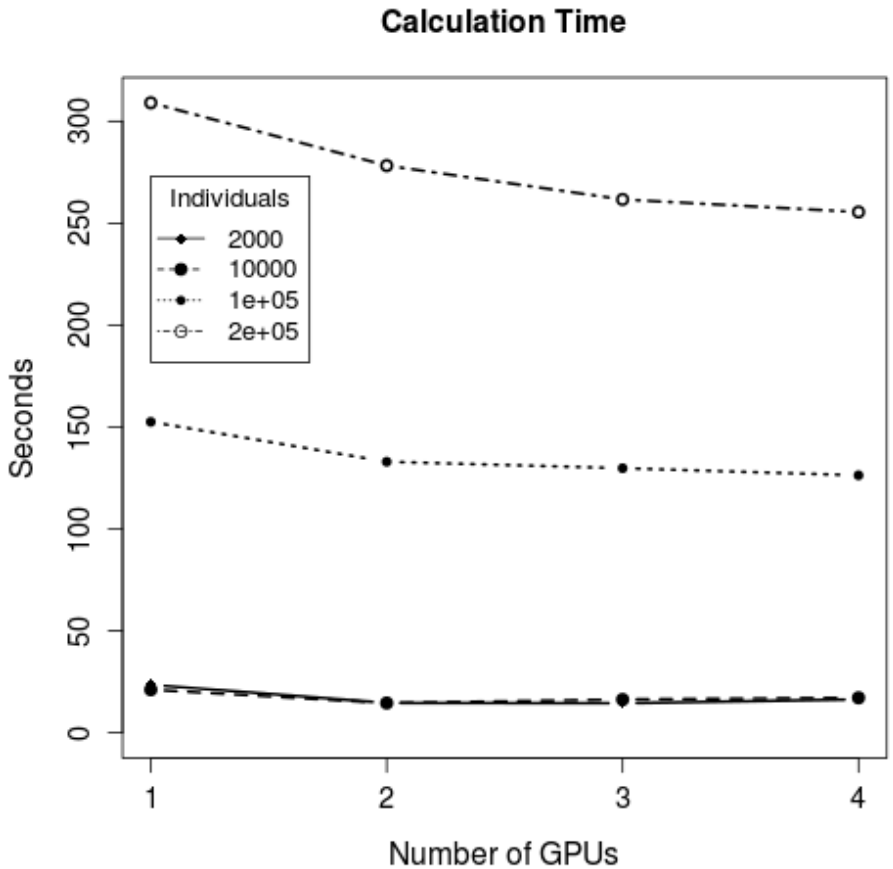
Data and Expectations

- Simulated data
- 10 000 SNPS
- 2000, 10 000, 100 000, 200 000 individuals
- 0, 5, 10, 20 covariates
- Expected linear efficiency

Streams


Calculation Time
Seconds
200
400
600
800
1000
2
4
6
8
10
Number of Streams
GPU
1
2
3
4

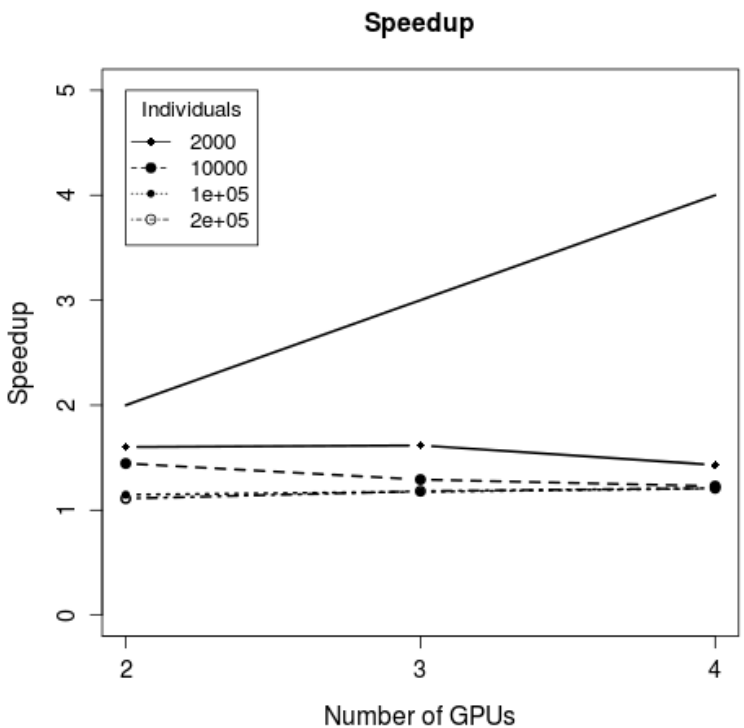
Saturated Streams



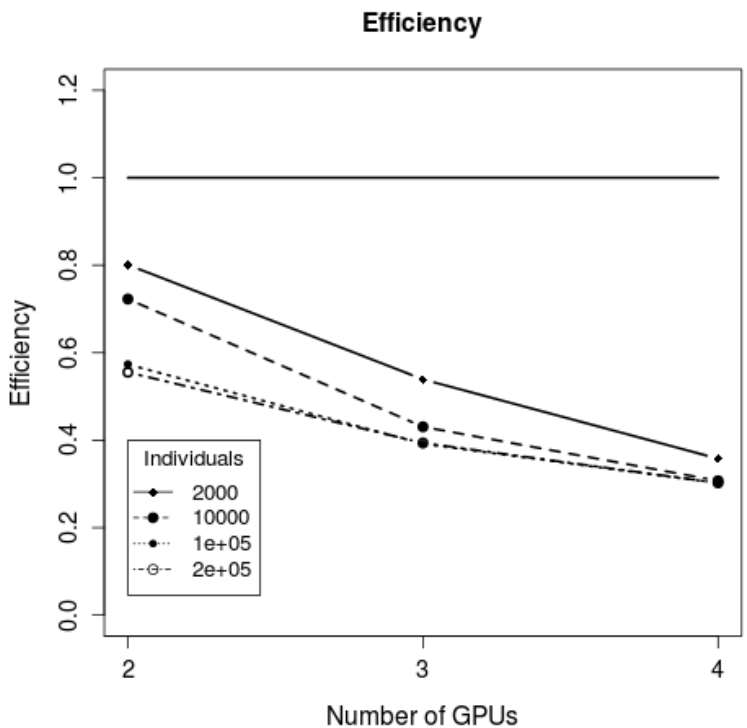
0 covariates

Speedup and Efficiency

Versus one GPU

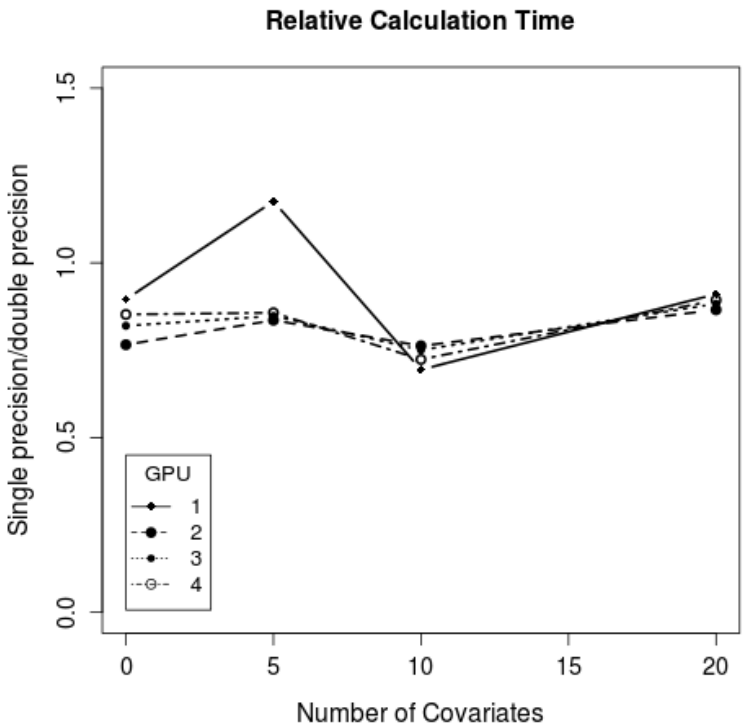


0 covariates

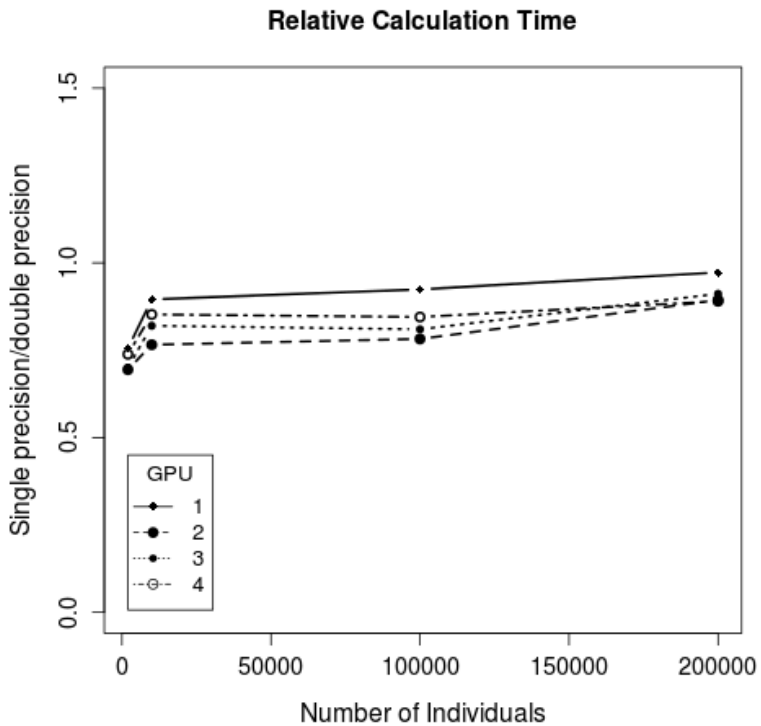


0 covariates

Single Versus Double Precision

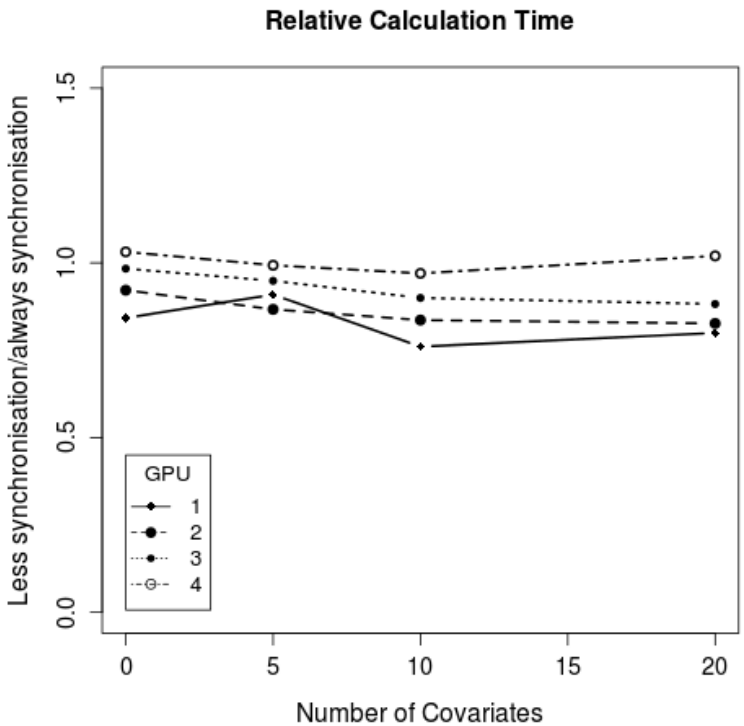


10000 individuals

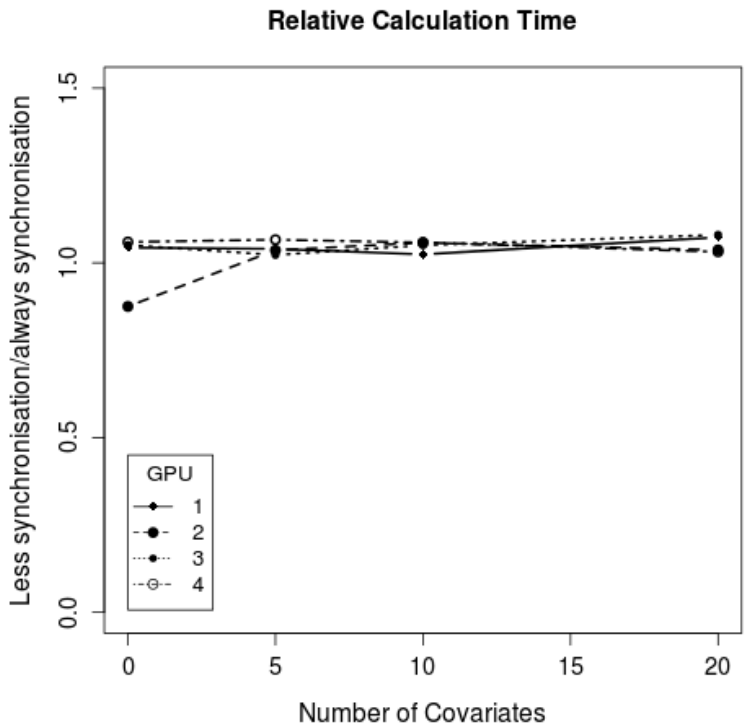


0 covariates

Synchronisation

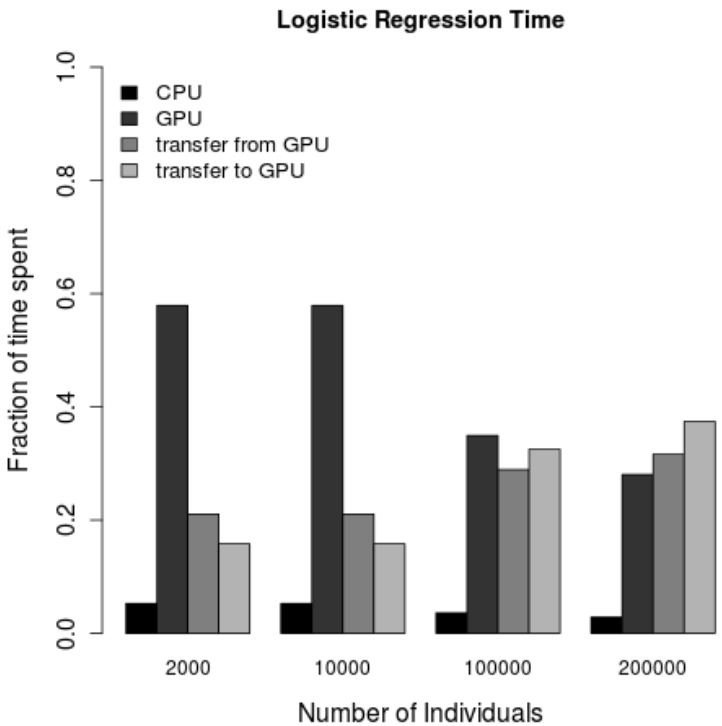


2000 individuals

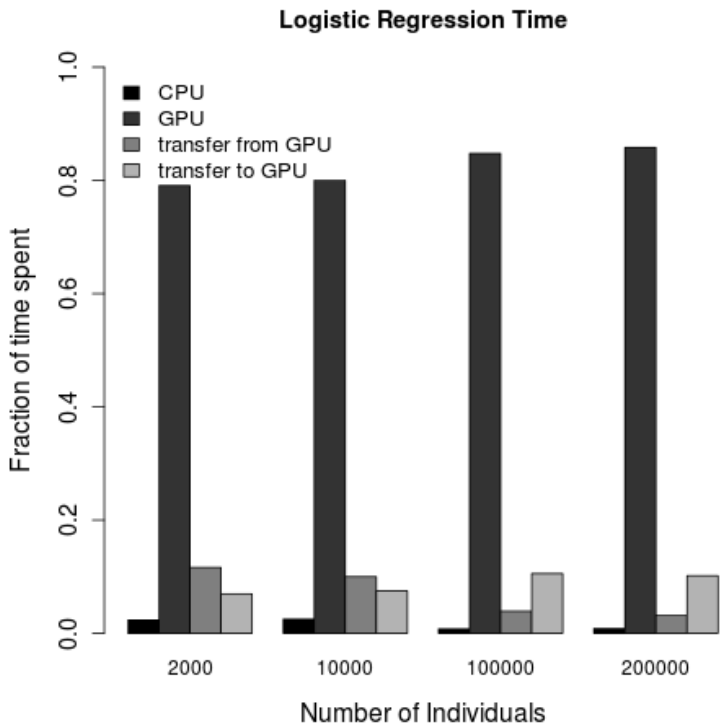


200 000 individuals

Synchronisation

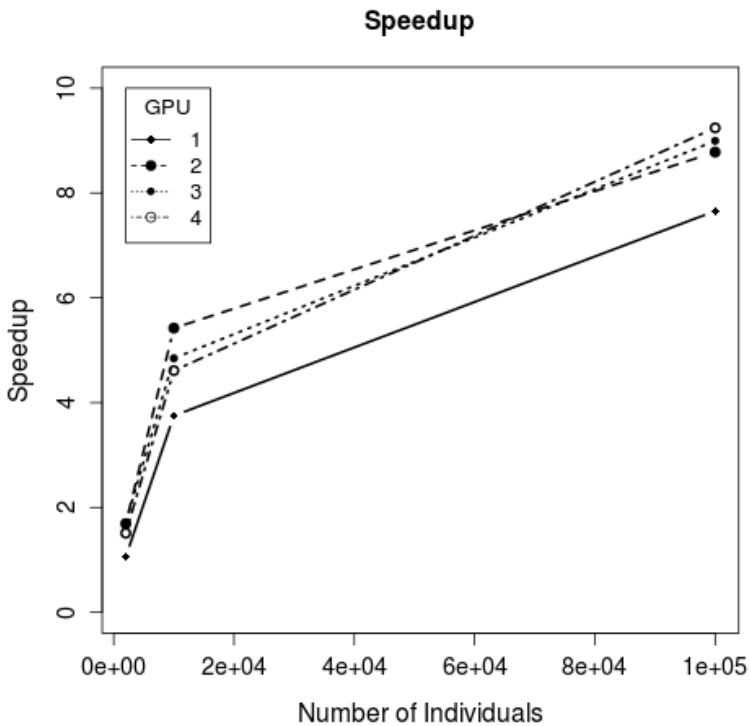


Less synchronisation

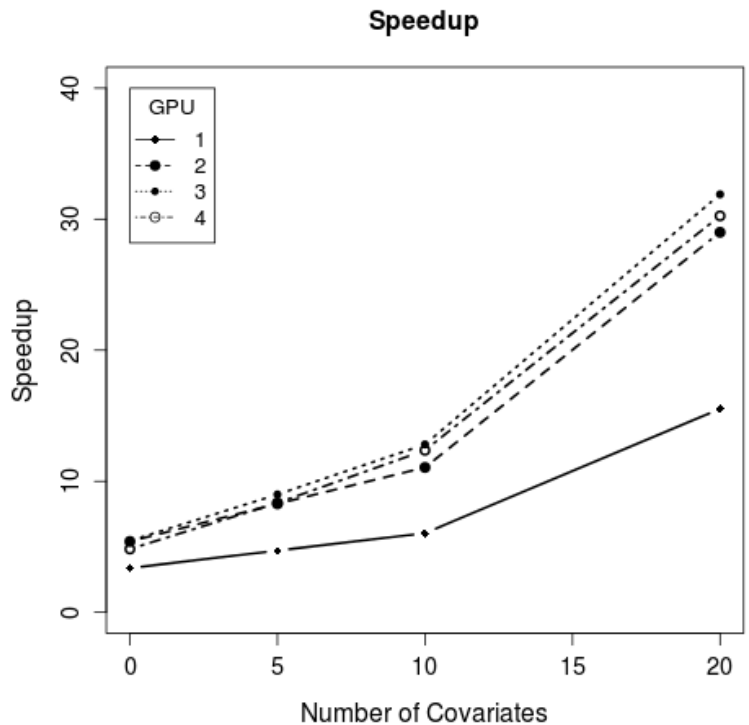


Always synchronise

Comparison versus GEISA



0 covariates



10 000 individuals

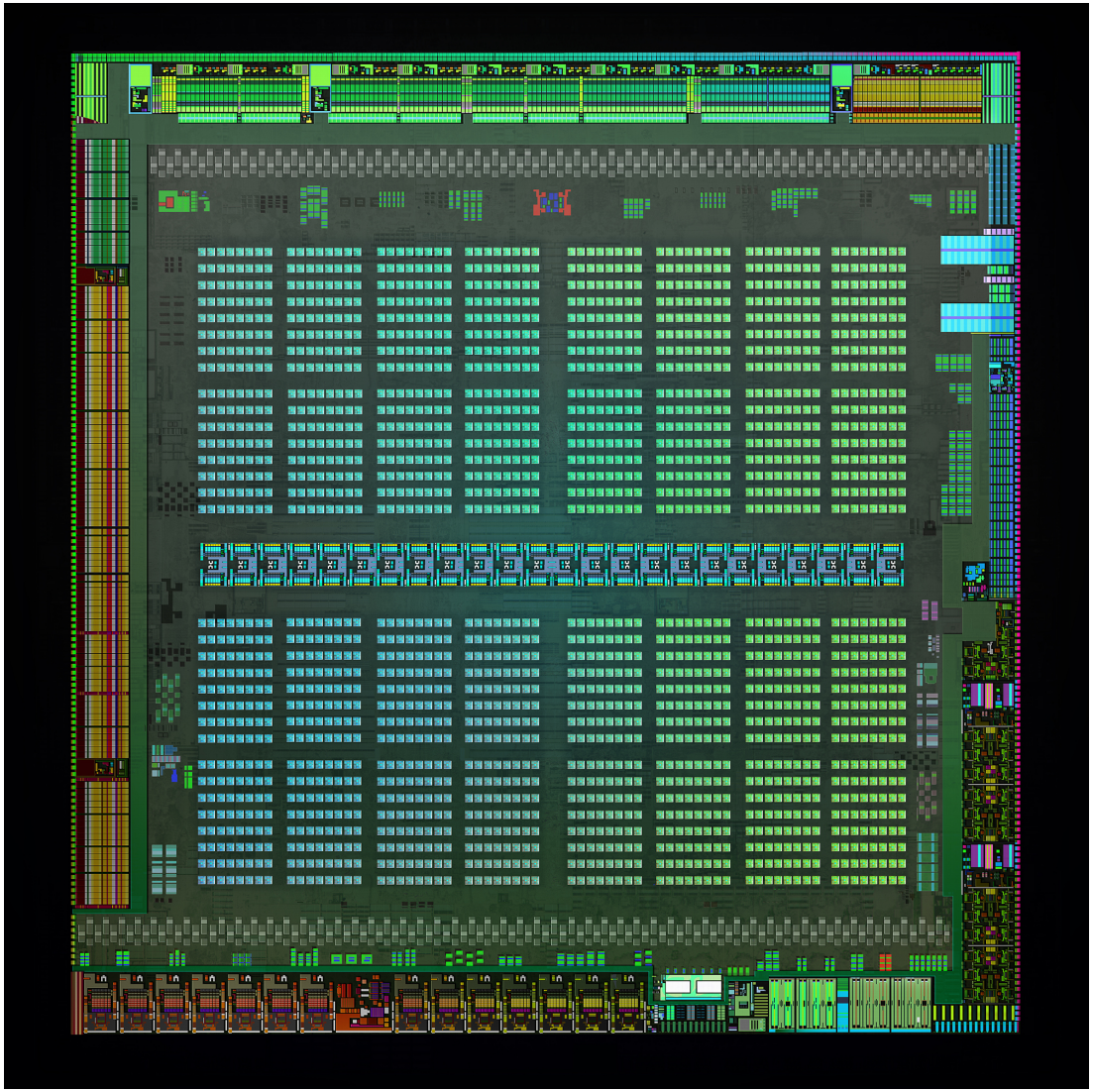
Conclusions

- GPUs suits well for interaction, depending on method
- Use one, perhaps two, GPUs with the program
- Single precision gives better performance than double precision
- Increased synchronisation increases performance slightly when number of individuals is above 100 000

Outlook

- Implement a method for validation, bootstrap or permutation tests
- Moving more parts to the GPU could fix the bad scaling
- Clusters for more speed
- Need for better statistics and definitions of interaction for non binary factors
- For binary factors look into gene-gene interaction methods if further speed is needed

GPU Architecture



CPU versus GPU

	CPU	MIC	GPU
Example	Intel i7-5960X	Xeon Phi 7120A	K80
Cores	8	61	4992
Memory	0.1-1 TB	16GB	24 GB
Caches	20 MB	30.5 MB	-
Concurrency	MIMD	MIMD	SIMT
FLOPS	0.4 T	3 T	3, 8.7 T
Price	$1000	$4200	$5000

Matrix Decomposition

- Pseudo inverse, A^+, is defined for general matrices
- Can be found by using singular value decomposition

$$A = U\Sigma V^T$$

$$A^+ = V\Sigma^+ U^T$$

Odds, Odds Ratio and Additive Interaction

$$\Omega = \frac{\pi}{1-\pi}$$

$$\theta = \frac{\Omega_1}{\Omega_2}$$

$$\theta_{\textit{both factors present}} > \theta_{\textit{first factor present}} + \theta_{\textit{second factor present}} - 1$$

Recoding

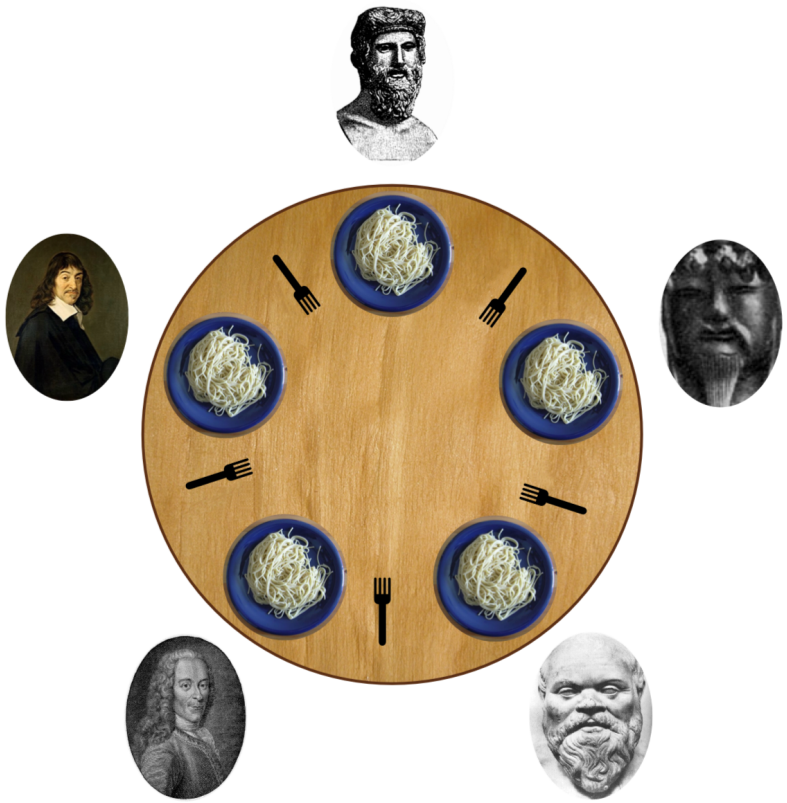
- The measures for additive interaction are defined for positive odds ratios
- Can be adjusted by recoding
- Recoding switches the reference group with the group with lowest risks
- Guarantees that $OR \geq 1$

Speedup and Efficiency

$$S(p) = \frac{T(1)}{T(p)}$$

$$E(p) = \frac{S(p)}{p} = \frac{T(1)}{pT(p)}$$

Concurrency and Dinning Philosophers



A Possible Solution

- Think
- Wait for a fork to become available and pick up that fork
- Wait for and pick up the other fork
- Eat
- Put down the forks one by one
- Go back to thinking

Deadlock

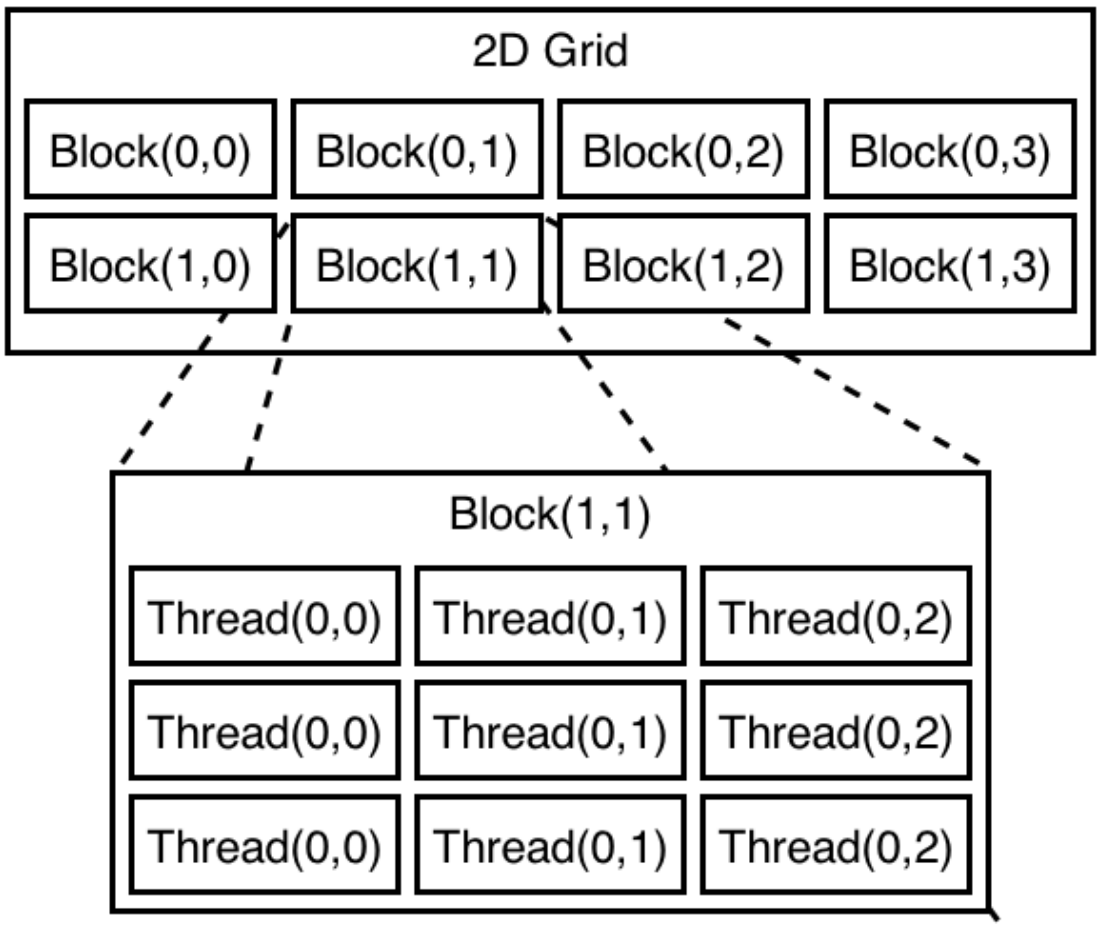
- Stuck when everyone holds one fork
- This is a deadlock
- Can be solved with a mediator

Time Distribution for Kernels

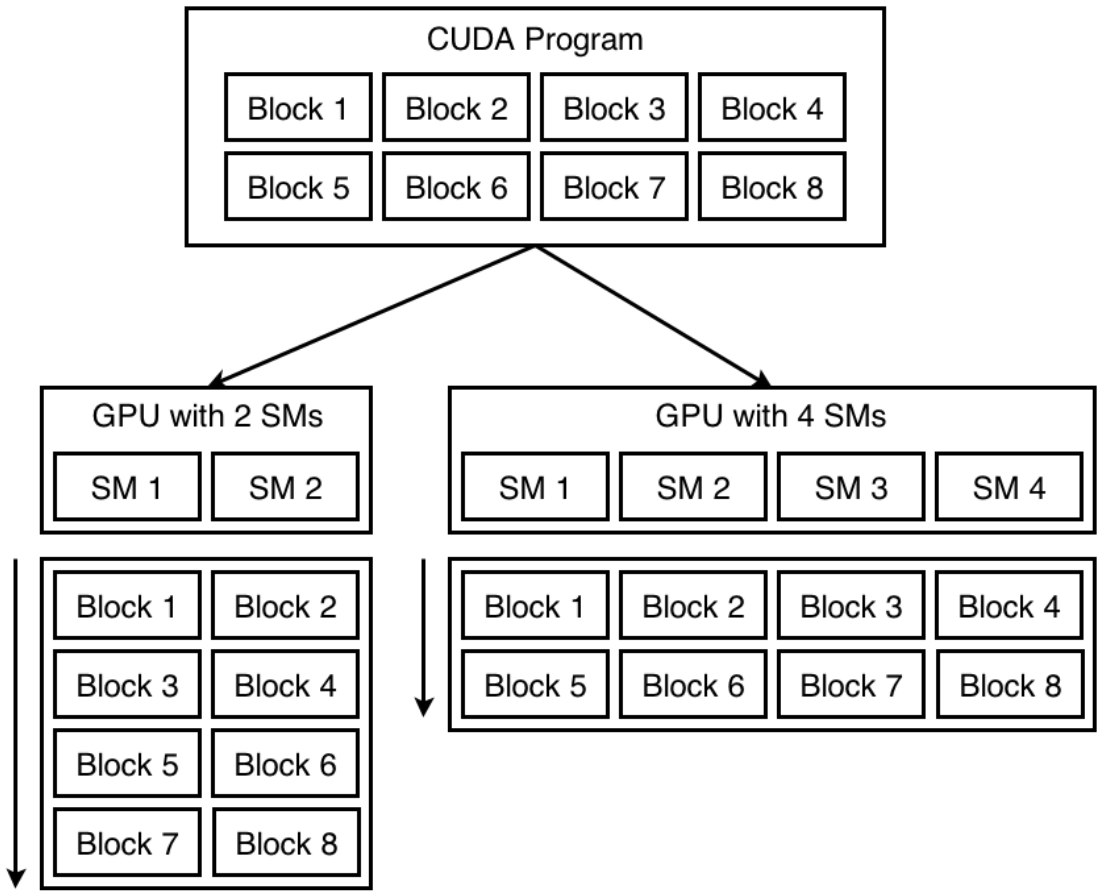
$*$ is element by element multiplication

Covariates Individuals		0 10k	0 100k	20 10k	20 100k
$M1^T \cdot V1$	CUBLAS	66.6	72.2	43	53.9
$M1 \cdot M2$	CUBLAS	18.9	14.4	36.1	25.8
$M1 \cdot V1$	CUBLAS	7.1	8.1	5.9	7.4
$V1 * V2$	Custom	2.9	2.3	11.8	10.6
$\frac{e^{V1}}{1+e^{V1}}$	Custom	1.1	0.9	0.7	0.7
$V1 * \log(V2) + (1 - V1) * \log(1 - V2)$	Custom	0.8	0.6	0.6	0.5
$V1 - V2$	Custom	0.7	0.6	0.5	0.4
$V1 * (1 - V1)$	Custom	0.7	0.5	0.4	0.4
Dot product	CUBLAS	0.7	0.3	0.3	0.2
$\Sigma V1$	CUBLAS	0.5	0.1	0.2	0.1

Blocks


2D Grid
Block(0,0)
Block(0,1)
Block(0,2)
Block(0,3)
Block(1,0)
Block(1,1)
Block(1,2)
Block(1,3)
Block(1,1)
Thread(0,0)
Thread(0,1)
Thread(0,2)
Thread(0,0)
Thread(0,1)
Thread(0,2)
Thread(0,0)
Thread(0,1)
Thread(0,2)

Blocks


CUDA Program
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
Block 8
GPU with 2 SMs
SM 1
SM 2
GPU with 4 SMs
SM 1
SM 2
SM 3
SM 4
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
Block 8
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
Block 8